REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Oct. 6, 1998	Final 6/16/97 - 6/15/98

4. TITLE AND SUBTITLE
Distributed Processing for Rapid Reconstruction of Terrain Models

5. FUNDING NUMBERS
DAAG55-97-1-0328

6. AUTHOR(S)
Howard J. Schultz and Edward M. Riseman

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
University of Massachusetts
Computer Science Department
Lederle Graduate Research Center
Amherst, MA 01003-4610

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park,, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 36897.1-MA-SDI

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. ABSTRACT *(Maximum 200 words)* The project goals are to develop high-performance computing techniques for rapid mapping of terrain and cultural sites to support a diverse set of capabilities for battlefield awareness. With the recent increase in the variety and resolution of image sources, there is an increasing need to rapidly generate specialized map products. To meet these requirements, the University of Massachusetts is developing parallel and distributed processing techniques. The first year research effort focused on mechanisms and strategies for using distributed processing techniques across a heterogeneous set of computer architectures, and parallel processing techniques for processing very large data sets. Several studies were conducted using the PVM (parallel virtual machine) library support. We successfully applied the technique to a complex perceptual grouping algorithm used in site model reconstruction. In addition, we began work on developing out-of-core algorithms that advantage multiprocessor servers to efficiently process very large data sets (images greater than 30,000 x 30,000 pixels). These algorithms circumvent the highly inefficient operating system virtual memory and paging algorithms. Initial bench match studies have shown that the efficiency of these out-of-core techniques is independent of the image size.

14. SUBJECT TERMS	15. NUMBER IF PAGES
distributed processing, parallel processing, site modeling, battlefield awareness	21
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Distributed Processing for Rapid Reconstruction of Terrain Models

June 16, 1997 – June 15, 1998

Howard J. Schultz. (PI)
Edward M. Riseman (Co-PI)

May 11, 1998

University of Massachusetts, Amherst

Contract Number DAAG55-97-1-0328

Approved for Public Release;
Distribution Unlimited

This is annual report covers the period June 16, 1997 – June 15, 1998 for ARO contract DAAG55-97-1-0328, "Distributed Processing for Rapid Reconstruction of Terrain Models".

The project goals are to develop high-performance computing techniques for rapid mapping of terrain and cultural sites to support a diverse set of capabilities for battlefield awareness. With the recent increase in the variety and resolution of image sources, there is an increasing need to rapidly generate specialized map products. To meet these demanding requirements we will develop parallel and distributive techniques.

The first year research effort focused on mechanisms and strategies for using distributed parallel processing across a heterogeneous set of processors, and parallel processing techniques for processing of very large data sets. We conducted two studies. The first (Appendix A) "Exploring PVM for a computer vision application" authored by Mr. Frank Stolle, a Ph.D. candidate in our research group, successfully demonstrated strategies for distributing computer vision applications across a network of heterogeneous computer architectures. The second "Processing Large Data Sets" describes a project to create large image manipulation routines and a bench mark to measure the performance of these routines.

The expanded scope of the research involved parallel processing techniques for handling very large data sets.

Processing large geospatial models.

Advances in the size, variety and speed of imaging sensors has resulted in an explosion of applications that utilize large geospatial databases. Generating, managing and visualizing these databases present unique computational problems that are not adequately addressed with current hardware and software systems. Increasingly, many of these issues are discussed in the scientific literature. Recently, several researchers have concluded that operating system virtual memory and paging algorithms are inadequate for interactive manipulation and visualization of large geometrically orientated databases.

We have also found this to be the case. For example, our current terrain modeling system (Terrest), which has been shown to be very robust and efficient, becomes impractical when the image size exceed 4,000 x 4,000 pixels Our investigations have shown that at runtime the internal intermediate representations requires about 100 times as much memory as is required to store a single image. Thus, if the input images are 16 Mbytes, approximately 2 Gbytes of physical memory might be required to generate the terrain model. Clearly, this scaling problem must be addressed if the terrain reconstruction techniques are to be extended to large image format, which are expected to exceed 30,000 x 30,000.

We have started a joint research program with Professors Chip Weems, Kathryn McKinley and Eliot Moss of the UMass Computer Science Department to develop software tools to manage memory hierarchies when geospatial data exceeds the limits of primary memory. These tools incorporate *out-of-core* disk-to-memory processing schemes optimized for single and

multiprocessor architectures and RAID systems (redundant array of independent disks). These procedures will be incorporated into our existing terrain modeling software. Our goal is to create a terrain modeling system capable of efficiently handling virtually any size input.

Publications:

Schultz, H., Jaynes, C., Marengoni, M., Schwickerath, A., Stolle, F., Wang, X., Hanson, A., Riseman, E., "3D Reconstruction of Topographic Objects at the University of Massachusetts", Joint ISPRS Workshop on 3D Recognition and Modeling of Topographic Objects, Stuttgart, Germany, September 17-19, 1997.

Schultz, H., Schwickerath, A., Stolle, F., Hanson, A., Riseman, E., "Incremental Digital Elevation Map Generation From Stereo Images", ISPRS Workshop on Theoretical and Practical Aspects of Surface Reconstruction and 3D Object Extraction, Haifa, Israel, September 9-11, 1997.

Schultz, H., Stolle, F., Wang, X., Riseman, E., and Hanson, A., "Recent Advances in 3D Reconstruction Techniques from Aerial Images", Proc. DARPA Image Understanding Workshop, New Orleans, LA, May 1997, Vol. II, pp. 977-982.

Appendix A

Exploring PVM for a Computer Vision Application

Frank Stolle
University of Massachusetts
Department of Computer Science
Amherst, MA 01003

May 26, 1998

1 Introduction

It is very common today for organizational units to have a number of networked workstations. Most of the time these machines are used locally and not to their fullest potential. A number of typical computer vision applications require significant processing power. Often powerful single processor or symmetric multiprocessor systems are used for these applications. The goal of this study is to investigate how practical it is to take advantage of distributed computing in a network of workstations to solve a computer vision problem. Two support libraries for distributed parallel computing are considered and a test application is implemented.

1.1 Parallel programming models and supporting libraries

The two main programming models for distributed processing are shared memory and message passing. The former exploits mechanisms to share memory between individual processors, the latter one makes use of explicit messages between different processing units. Mixed forms like distributed shared memory are being used to keep local copies of the memory in each machine, and use message passing to keep the individual copies consistent.

In this project, the message passing paradigm for parallel programming was chosen because of its greater simplicity. There are libraries available that take the burden of writing the message passing routines away from the application designer.

1.2 Libraries for message passing: PVM and MPI

PVM (Parallel Virtual Machine) [18, 19] is a software tool to support distributed computing based on a message-passing paradigm. It allows the user to join machines of different architecture interconnected by various possible networks into a single virtual parallel machine. The hardware supported ranges from PCs to massive parallel machines [22].

PVM supports operations for adding and deleting physical machines into and from the virtual machine, synchronization operations, and message encoding for exchange between different architectures. PVM supports the programming languages C and FORTRAN. Several visualization tools exist for studying the process behavior and the network traffic.

MPI (Message Passing Interface) [20] was developed in the early 90s in an effort to combine features of previously existing message passing systems. It has been influenced by work at the IBM T. J. Watson Research Center [14, 15], Intel Corporation and PVM [16, 17]. The MPI standardization effort involved approximately forty organizations from the United States and Europe. MPI offers support for point-to-point communication, collective communication, process groups, management of process environments and process topologies, and provides an easy-to-use C and FORTRAN library interface. One of the goals of MPI is to provide MPP vendors with a clearly defined base set of routines that they can implement efficiently.

The experiments described throughout this paper made use of the PVM library. PVM was chosen since it focuses more on the concept of a virtual machine, was widely available and provided the necessary functionality.

2 Hardware

The UMass computer vision lab has a set of different machines that allows interesting experiments with a system such as PVM. Experiments were carried out with a set of SUNs and SGIs, connected by standard thinwire ethernet.

Table 1 shows information regarding each type of machines used in the experiments.

	Machines				
Description	Escher	Periscope	Helios	Colossus	Jenni
Number of Processors	2	4	1	1	1
Processor Type	MIPS R4400	MIPS R2000/R3000	SuperSPARC	SuperSPARC	MicroSPARC
Speed (MHz)	200	33	50	50	50
Cache Size (Kbytes)	16	64	16	16	4
Secondary cache (kbytes)	4000	256	1000	1000	na
Main memory (Mbytes)	64	128	64	96	32

Table 1: This table shows the characteristics of the machines used in this project.

2.1 The application

The University of Massachusetts computer Vision Group has been developing a system for model reconstruction from aerial imagery [1, 6]. It exploits various techniques of perceptual grouping [9] to obtain higher-level information. A similar approach is taken in several other image understanding systems, e.g. [7].

The initial scheme of the system involved finding building rooftops using graph algorithms. The idea is to recognize closed rectilinear cycles, most of which correspond to rooftops with high probability. A feature-relation graph is constructed from lines extracted from the image. Line intersections yield vertices in the graph, and vertices are connected by an edge in the graph when there is enough line coverage in the image between the two vertex coordinates. This data is processed in subsequent steps to find closed polygons.

Input for the algorithm is a set of line segments generated by a line finder algorithm.

The line input from the line finder is used to generate corner features. Lines of a certain minimal length and a certain minimum and maximum distance are checked for intersection. For the intersection check lines are extended in length along the original direction in order to find intersections when lines are very close together, but do not actually intersect. The generated corners carry a weight and two orientation parameters (the orientations of the lines they were generated from)

More corner features are hypothesized from lines exceeding a certain length. These corners only carry a weight, but have no defined orientations.

Redundant corners from both corner generation steps are then merged to reduce the number of corner features. The weight of a resulting corner will be a weighted average of the weights of the corners it was generated from. Its orientations will either be defined, if the original corners had the same orientations within a certain range, or they will be not defined. Dihedral corners in the image will have defined orientation parameters, corners with more legs will not.

The edge generation step checks for line support in between any two corners which have a certain, image dependent, minimum and maximum distance.

Corners with defined orientation parameters can only be linked with other corners if the line connecting the two lays within a certain range of angular deviation from the direction vectors specifying the orientations of the legs of the corner. Corners without orientation constraints can be linked with any other corner given the other conditions are met.

2.2 Master-slave paradigm

The first key decision on this algorithm was to choose the master-slave paradigm. A master process schedules the work for a number of slave processes. The slaves send results back to the master which does all processing that should be centralized and then sends out work requests to the slaves. There are two operations a slave can carry out: compute corners from line segments and check for line segment support between corners. Every machine has a file with the initial line segments on a local disk. When the program starts, slaves read the data and the master sends range information to each slave specifying the data range the slave should compute corners from. When the slave is done it sends back the corner information (the location and weight of each corner) to the master. The master continues to send requests until every possible combination of line segments has been checked. After all corners have been detected, the master generates a new list that contains fewer corners by deleting corners that are close in proximity and replacing them with a single corner. The list of corners after that operation is sent to every slave. The master again sends range information to each slave specifying the range each slave will have to compute edge support for. After a slave is done it sends computed edges between corners back to the master. The master joins the individual lists and writes the graph out to file. This file is used in a later stage by a graph matching algorithm to generate polygons.

3 Experiments

Three different sets of experiments were carried out and will be described below. The first one used PVM in transparent mode (PVM does the task scheduling) with one task per slave and varying numbers of slaves. The second set used a fixed number of slaves while the tasks were sub-partitioned into smaller parts to be able to use more than one sub-task per slave. The third and last set used explicit task scheduling (an explicit number of slaves are started on each machine) with a fixed number of slaves and a varying number of sub-partitions per task.

3.1 Experiments with different configurations of the virtual machine

The first set of experiments used the same input data but different configurations of the virtual machine and different task partitioning. The input data was the output of Boldt's algorithm on the image of a building on Kirtland Air force Base. It contained 1488 line segments. Individual experiments were carried out five times to minimize random errors.

The following important numbers are reported in a table for each experiment:

- Master total time - the total time between start of the master and finishing to compute the result
- Master work time - the time the master spends on computation
- Master overhead time - the time the master spends on communication (including packing and unpacking of data) and waiting
- Average slave work time - the average time a slave spends on computation
- Average slave overhead time - the average time a slave spends on communication (including packing and unpacking) and waiting

Table 2 illustrates the different configurations used.

Configuration	Iris	Helios	Jenni	Colossus	Escher	Periscope
V1	X					
V2			X			
V3				X		
V4					X	
V5						X
V6					X	X
V7	X	X				
V8	X	X	X	X		
V9	X	X	X	X	X	X

Table 2: Configurations of the virtual machines for the experiments

It should be noted that configurations V1 - V5 used a single machine only. These configurations were mainly used for comparison against multi-machine configurations and to explore the behavior of multi-processor systems (V4 and V5) with PVM.

As mentioned above, three sets of experiments were carried out. The first one used PVM in transparent mode (PVM does the task scheduling) with one task per slave and varying numbers of slaves. In these experiments the task was split into equal size sub-tasks and each slave was assigned a sub-task to work on. For example, with four slaves each of them checked one quarter of the line segments against the full amount of line segments for overlap. After corner merging by the master each slave was assigned an equally sized range of corners to check for edges in between that range and the full corner range. This scheduling mechanism was not optimal as the results show. The issue was addressed in the other sets of experiments.

The second set used a fixed number of slaves. The tasks were sub-partitioned into smaller parts to be able to use more than one sub-task per slave.

The third and last set used an explicit scheduling mechanism (an explicit number of slaves on each machine) with a fixed number of slaves and a varying number of sub-partitions per task.

Figures are included to visualize results.

3.2 Experiments with a transparent mode, one task per slave

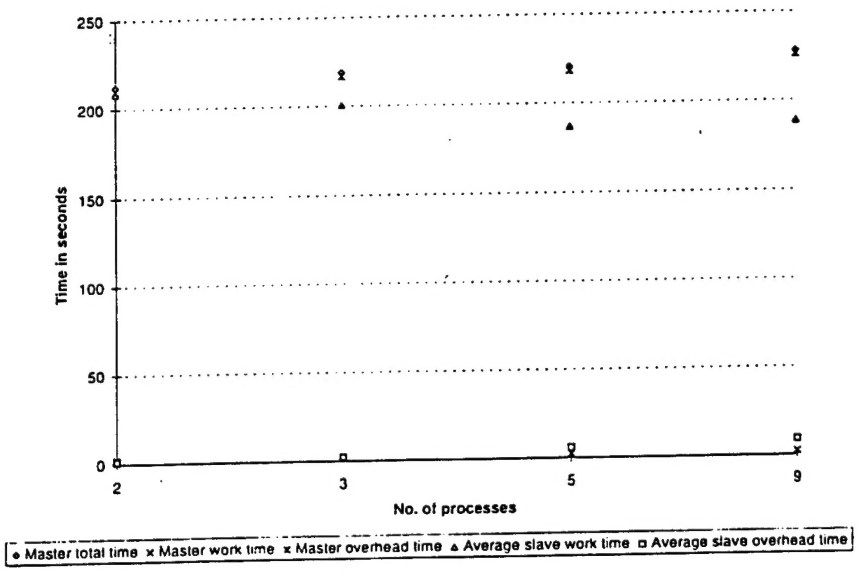

Figure 1: Time versus number of processes for configuration V1

The results of the runs on configuration V1 showed a slight improvement of the average slave work time as the number of slaves increased up to 4 slaves. Concurrently time slave overhead increased steadily. With more slaves each individual slave has to do less operations. Increasing the number of slaves in this configuration should not improve the speed since there is only one CPU to do the work and the slaves processes have to share it. This should in fact lead to a increase in total slave time when scheduling multiple slaves on the same system. The initial slight improvement in slave work time we actually see may be due to process management issues in the operating system or caching.

Configuration V4 utilized a two-processor SGI. As the results show, the speedup when using both processors on this machine with a single slave was at a rate of about 1.75. The best possible speedup could be achieved with 4 slaves on each processor. While the slave overhead time steadily increased, the average slave work time has a minimum at that point. As mentioned above, this may be caused by operating system process scheduling and caching strategies on the single machine.

As expected the results show a small fraction of time of the master process devoted to computation while the majority is overhead. Most of the master overhead is spent waiting for the results from slaves. The slaves in turn spend most of their time computing and have a small overhead.

The different physical machines need different amounts of time to finish the program with one slave using the same data.

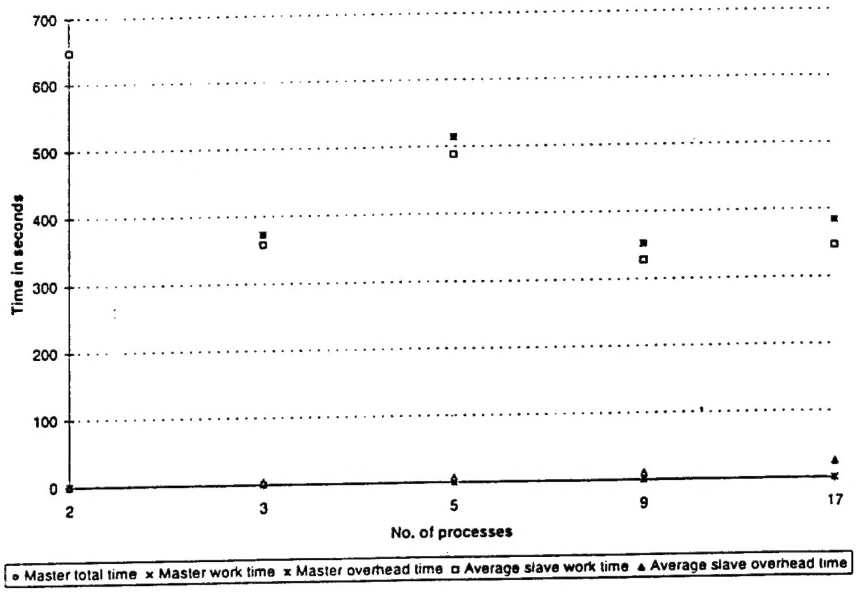

Figure 2: Time versus number of processes for configuration V4

The performance for the systems with more than one processor improved as more processors are used. V4 showed a speedup of about 1.75 when two instead of one slaves are used. V5 performed about 3.45 times as fast when all four instead of only one processors were used.

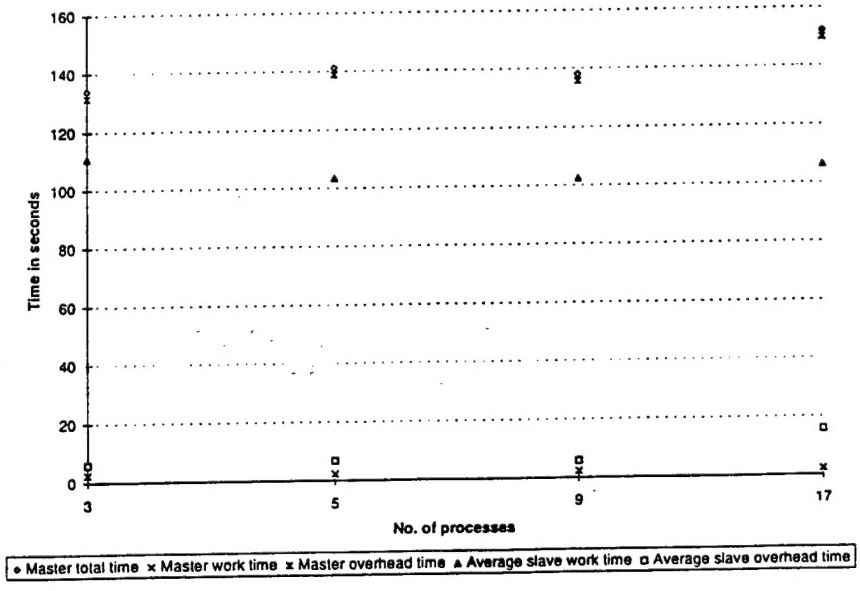

Figure 3: Time versus number of processes for configuration V7

Configuration V7 utilized two SUN Sparc stations. The average slave work time decreased between 1 and 4 slaves per machine and went up as the number of slaves was increased further. The slave overhead time increased steadily as more slaves were used.

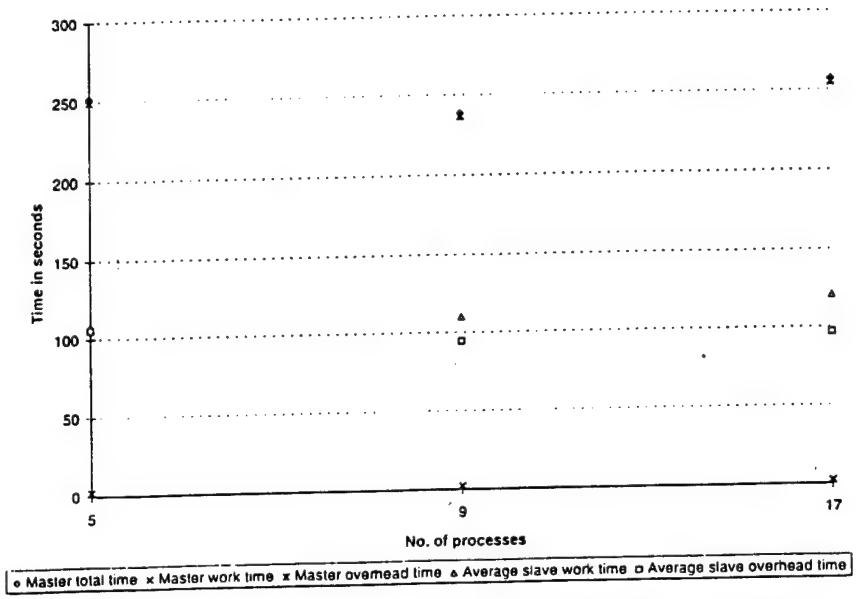

Figure 4: Time versus number of processes for configuration V8

Configuration V8 utilized all four SUNs. The average processing time per slave was about the same as for configuration V7, but the average slave overhead has gone up to about the same value. This was caused by waiting for the slowest machine to finish tasks. The optimum for this configuration was 2 slaves per machine. The lowered overhead time outweighed the increased work time per slave. In this configuration it would be beneficial to break up the tasks into smaller pieces and not to assign each slave an equal size problem. Later experiments described below address this issue. Due to the bad scheduling V8 has a lower performance than V7 in this mode.

Configuration V9 showed a speedup between 6 and 24 slaves used. This was mainly due to the speedup on the slowest physical machine that could use all of its four processors with a total of 24 slaves in this configuration. It should be noted that without explicit scheduling or sub-partitioning the tasks V9 had a lower performance than V7 despite the fact that it contained all the physical machines included in V7.

3.3 Experiments with different task granularity / message sizes

Another issue to be investigated was dependency of the effect of changing granularity of task partitioning. Finer granularity allows for more flexibility in task scheduling since it can allow slower machines to do less work and faster machines to do more. A finer granularity usually results in more messages of smaller size, coarser granularity yields less but larger messages. Smaller messages can be delivered faster but there will be more messages and more overhead to transmit them.

Dividing the task into smaller pieces showed the expected effect on the single machine, single processor configuration V1. Both slave work time and slave overhead time increased as the number of sub-tasks increased. With only one processor dividing the work into smaller sub-tasks should not improve speed. All slaves had roughly the same performance since scheduling was done by the operating system.

Configuration V4 did not show a benefit from smaller task partitioning with one slave running on each processor. Since both processors had the same characteristics they needed about the same

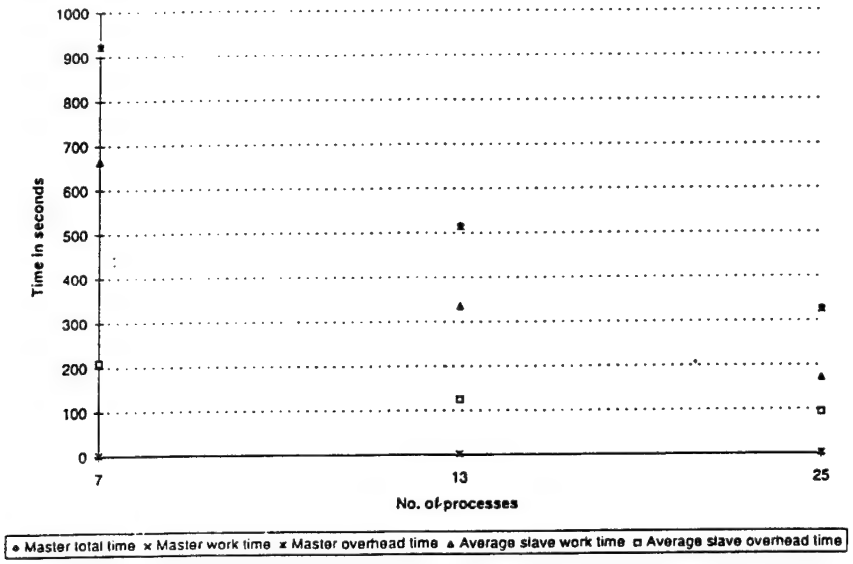

Figure 5: Time versus number of processes for configuration V9

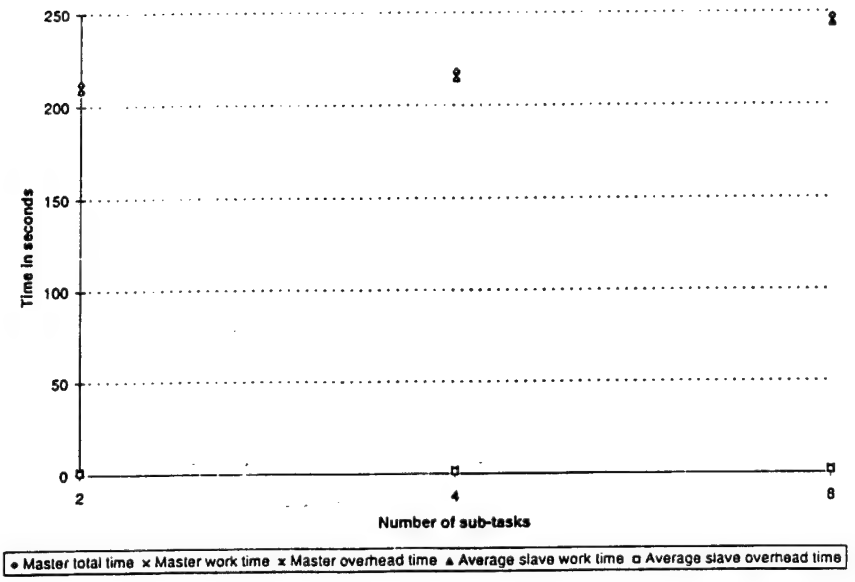

Figure 6: Time versus number of sub-tasks for configuration V1 with 2 slaves

amount of time for the same task. Therefore no excessive overhead times for waiting occurred. There was no gain in smaller partitions and the increased overhead caused a slowdown with smaller sub-tasks.

Configuration V7 consisted of two SUN Sparc stations as used in V1. When the number of sub-tasks increased an initial decrease in total time could be noticed. This was mostly due to a decrease in slave overhead time. Since individual slaves did not have to wait as long for other slaves

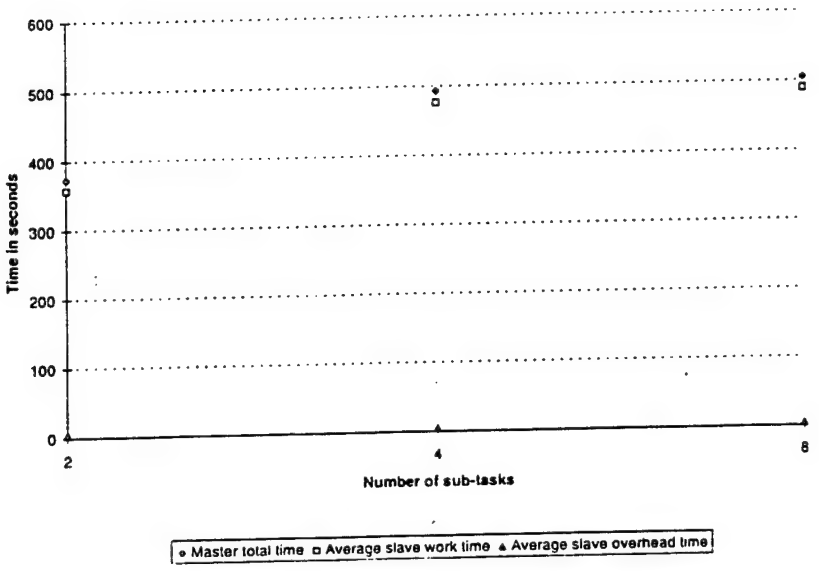

Figure 7: Time versus number of sub-tasks for configuration V4 with 2 slaves

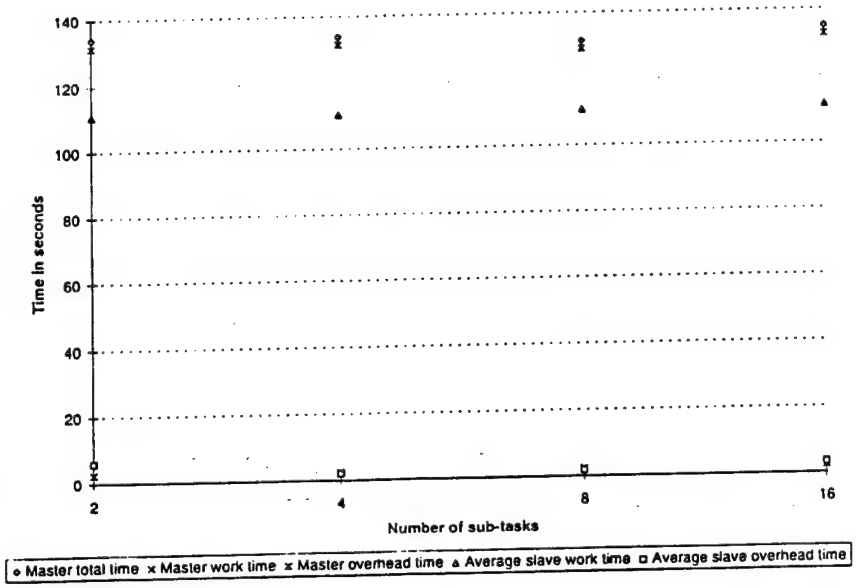

Figure 8: Time versus number of sub-tasks for configuration V7 with 2 slaves

to finish with smaller sub-tasks overhead time could be reduced. With more than 8 slaves the slave overhead time increased again due to more communication.

V8 consist ed of four SUN machines. Its best performance could be achieved with 8 sub-tasks, that is two per processor. Figure 9 shows a larger increase in total time between 8 and 16 sub-tasks than one could expect from the increase in average slave work time. The reason was mainly increased master overhead time due to increased communication.

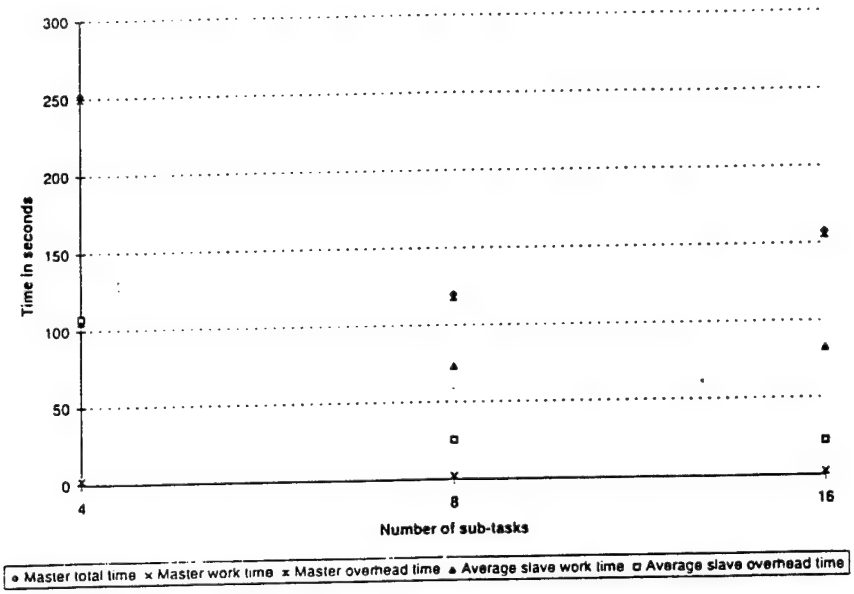

Figure 9: Time versus number of sub-tasks for configuration V8 with 4 slaves

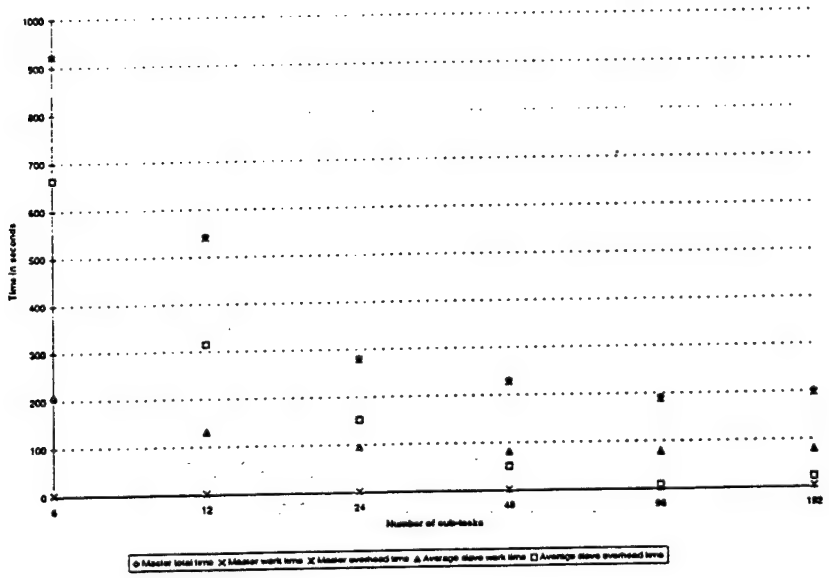

Figure 10: Time versus number of sub-tasks for configuration V9 with 6 slaves

Configuration V9 used all available machines. With 6 slaves its best overall time could be achieved between 160 and 320 sub-tasks. Figure 10 shows that increased scheduling flexibility with more sub-tasks initially improved both average slave work time and slave overhead time. After a maximum speedup had been reached further increase in the number of sub-tasks yielded a worse result when the communication overhead became dominant.

Figure 11 shows the different behavior of a multi-processor SGI with 2 processors (V4) versus

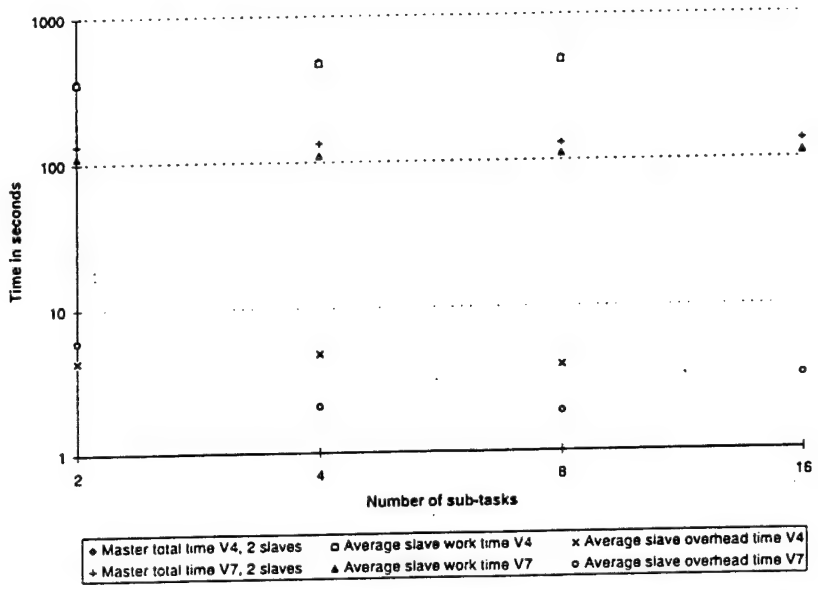

Figure 11: Time versus number of sub-tasks for configurations V4 and V7

a virtual machine with two SUN Sparc processors (V7). On V4 an increase in the number of sub-tasks increased the total time needed. On V7 an increase in the number of sub-tasks from 2 to 8 slightly decreased the total time needed since task scheduling could be done more efficiently. Since the two SUNs were running individual operating systems with individual process schedulers this had a larger impact than on V4 that ran a single operating system and had tightly coupled processors. The scheduling outweighs the increase in overhead for V7 up to 8 sub-tasks.

3.4 Experiments involving explicit task scheduling

The third set of experiments carried out was to investigate what effect manual task scheduling would have on the performance. The previously described experiments were all carried out in transparent mode. In this mode PVM itself can choose on which physical machines to start individual slaves. It usually tries to assign an equal number of tasks to each physical machine. If machines have different performances then this behavior will often not yield the best performance. As seen in the previous set of experiments one can use finer task partitioning. This way faster machines will be able to do more work. However, for a virtual machine containing physical machines with different numbers of processors scheduling should be adjusted. Each machine should run at least one slave per processor that is available for this application. PVM provides a mechanism for starting tasks on specific physical machines. For the experiment in this section, one slave process was run per processor in each machine in the full configuration. Thus a total of 10 slave processes were used in each run.

Configuration V9 with explicit scheduling and sub-partitioning yielded the best overall performance. Without sub-partitioning the slave overhead time determined the total time needed since physical machines of different characteristics were used. As the task granularity became finer the performance increased until an optimum was reached with 320 sub-tasks total. The slave work time decreased quickly between 10 and 80 sub-tasks. With more than 320 sub-tasks (that is about 32 per slave) it increased again. Slave overhead time showed about the same behavior but changed

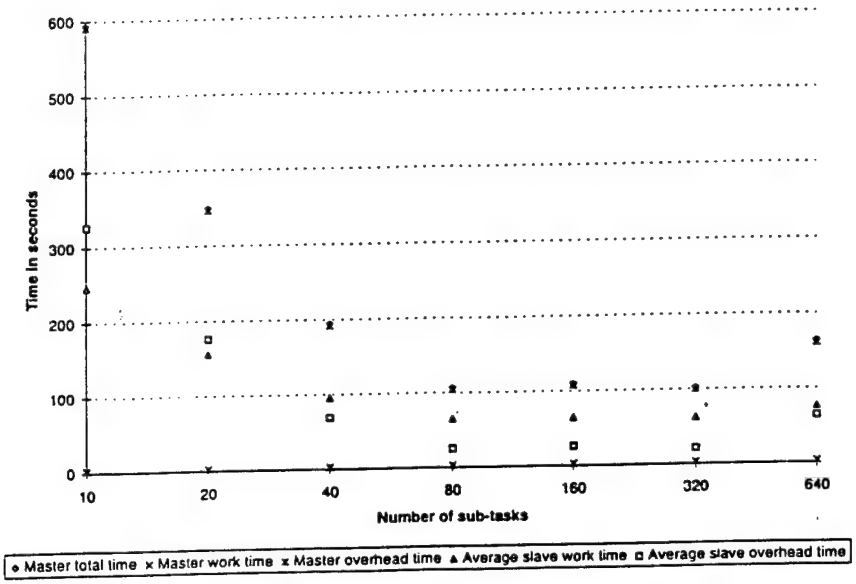

Figure 12: Time versus number of sub-task for configuration V9 with 10 slaves and explicit task scheduling

more rapidly. The decrease in slave work time was mostly due to faster machines getting more workload and therefore a better overall slave performance.

4 Conclusions

The general conclusions after this set of experiments are the following:

1. It is very important to have an intelligent load scheduling mechanism if the virtual machine consists of individual machines with different characteristics. If the same workload is send to each machine regardless of its specifications the overall speed is determined by the slowest machine. If machine performance is known beforehand the program can divide the workload accordingly. If machine performance varies due to other tasks or is not known a priori dynamic load scheduling should be used. Support for dynamic load scheduling could be included into PVM to make it more effective.

2. For the case where a machine in the virtual machine has more than one processor the *transparent mode* is not really adequate because PVM does not recognize this fact. The only way to by pass this in the *transparent mode* is to divide the overall job into a number of tasks which is a multiple of the number of processors in the parallel machine.

3. PVM performs well in the *transparent mode* if the machines involved in the virtual machine are all single process machines with similar speed.

4. Careful consideration has to be given which machine to run the master process on. A dedicated machine might be necessary in case of heavy workload for the master. In turn, if the master processes does not require much computational power, it could be run on the machine with the lowest performance to free up resources for slaves.

5. The message size had no direct significant influence in the overall performance of the machines for the application used in this experiment, but the number of messages in the network had a large influence in the system performance.

6. There is an optimal number of processes running on a single machine. Increasing the number of slaves on a single processor to be larger than one may increase the performance. An even larger number of slave processes on a single processor will decrease it again. This effect may be caused by the operating system limiting the amount of CPU time a single process can use.

7. It is very important to partition the tasks such that the master process doesn't have to wait for one or more slower slaves while all the faster slaves are idle.

8. It could be shown that the perceptual grouping application used is well suited for parallel speedup in a master-slave paradigm.

References

[1] R. Collins, Y-Q. Cheng, C. Jaynes, F. Stolle, X. Wang, A. Hanson, R. Riseman. "Site model acquisition and Extension from Aerial Images", *Fifth International Conference on Computer Vision*, pp. 888-893, 1995

[2] T. Cormen, C. Leiserson, R. Rivest. "Introduction to Algorithms", *The MIT Press*, pp. 600-604, 1992

[3] Wolfgang Förstner, "A Framework for Low Level Feature Extraction", *European Conference on Computer Vision*, 1994, pp 383-394

[4] Michael R. Garey, David S. Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness", *W. H. Freeman and Company*, 1979

[5] M. Grötschel, L. Lováz, A. Schrijver, "Geometric Algorithms and Combinatorial Optimization", second edition, *Springer-Verlag*, 1994

[6] C. Jaynes, F. Stolle, R. Collins. "Task Driven Perceptual Organization for Extraction of Rooftop Polygons", *IEEE Workshop on Applications of Computer Vision*, 1994

[7] Claudia Fuchs, Wolfgang Förstner, "Polymorphic Grouping for Image Segmentation", *International Conference on Computer Vision*, 1995, pp. 175-182

[8] D. Kozen. "The Design and Implementation of Algorithms", *Springer-Verlag*, pp. 101-110, 1992.

[9] D. Lowe, "Perceptual Organization and Visual Recognition", *Kluwer Academic Publishers*, 1985

[10] C. Papadimitriou. "Computational Complexity", *Addison-Wesley Publishing Company* pp. 188-190, 1994.

[11] C. Papadimitriou. "Computational Complexity", *Addison-Wesley Publishing Company* pp. 307-309, 1994.

[12] C. Papadimitriou, K. Steiglitz. "Combinatorial Optimization: algorithms and complexity" *Prentice Hall*, pp. 454-481, 1982.

[13] Dana Richards, Arthur Liestman, "Finding cycles of a given length" *Annals of Discrete Mathematics, Elsevier Science Publishers B.V* pp. 249-256, 1985.

[14] V. Bala and S. Kipnis, "Process groups: a mechanism for the coordination of and communication among processes in the Venus collective communication library", Technical report, IBM T. J. Watson Research Center, October 1992.

[15] V. Bala, S. Kipnis, L. Rudolph, and Marc Snir, "Designing efficient, scalable, and portable collective communication libraries" Technical report, IBM T. J. Watson Research Center, October 1992.

[16] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, "Visualization and debugging in a heterogeneous environment", IEEE Computer, 26(6):88–95, June 1993

[17] J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, "Integrated PVM framework supports heterogeneous network computing", Computers in Physics, 7(2):166–75, April 1993.

[18] Adam Beguelin, J. Dongarra, Al Geist, Robert Manchek, Keith Moore, and Vaidy Sunderam. Tools for Heterogeneous Network Computing, pp 854-861, Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Ed. R. Sincovec et al. 1993, SIAM Publications, Philadelphia.

[19] V. Sunderam, J. Dongarra, A. Geist, and R Manchek, The PVM Concurrent Computing System: Evolution, Experiences, and Trends, Parallel Computing, Vol. 20, No. 4, April 1994, pp 531-547.

[20] The Message Passing Interface, International Journal of Supercomputing Applications Volume 8 Number 3/4, Fall/Winter 1994 (updated 5/95).

[21] J. Dongarra and Peter Newton, Overview of VPE: A Visual Environment for Message-Passing Parallel Programming, Heterogeneous Computing Workshop '95, Proceedings of the 4th Heterogeneous Computing Workshop, Santa Barbara, CA, April 25, 1995.

[22] Henri Casanova, Jack Dongarra, and Weicheng Jiang, The Performance of PVM on MPP Systems, University of Tennessee Technical Report CS-95-301, August 1995.

[23] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra, MPI: The Complete Reference, MIT Press, 1995.

[24] Robert L Pennington, "Distributed and Heterogeneous Computing", Cluster Computing Lecture Series by Robert Pennington, Pittsburgh Supercomputing Center at Southampton University, 1995.

Appendix B

Processing Large Data Sets

The large image benchmark is designed to compare the performance of computer system architectures when used to process large images. A large image is one that cannot fit comfortably in the real memory of a computer. Current image processing needs require the ability to process images with upwards of 8k by 8k (67,108,864) pixels. The predominant aspect of the benchmark is the transfer of image "chunks" into and out of real memory. The benchmark measures elapsed time and data transfers on simple tasks such as a 1D Gaussian and an image transpose (i.e., NxM => MxN).

The benchmark consists of two parts. The ALGORITHM part is a set of image processing tasks that access image files through a defined application interface. The ACCESS part consists of an image access and buffering system that implements the application interface. The application interface is designed so that knowledge about the order in which pixels will be accessed can be communicated from the ALGORITHM part to the ACCESS part. This way, the ACCESS part may be modified to utilize this information while still maintaining the basic image processing algorithms in the top part.

We currently have three variations on the ACCESS part. Two of these variations have been built and tested. Result timings have been obtained. The third variation is under construction. Users of the benchmark can supply their own variation as long as the application interface is not modified. Both the ALGORITHM part and the ACCESS part are supplied in source as part of the benchmark. The source consists of C code that is ANSI C and POSIX 1003 compliant.

The first variation (SYNCHRONOUS) on the ALGORITHM layer implements synchronous file data transfers. It does not benefit in any way from knowing in advance the order in which the pixels will be accessed. The second variation (THREADED) uses POSIX pthreads to implement overlapped synchronous file data transfers. It relies upon multi-tasking with light-weight threads and semaphores. The third variation (ASYNCHRONOUS) uses the new POSIX asynchronous IO protocol.

The THREADED version uses one thread to process the image, one thread to read the image, and one thread to write the result. The SYNCHRONOUS version uses a single thread to do all three operations. The amount of computation per pixel of an image is relatively small even for the 1D Gaussian.

We have compared the SYNCHRONOUS and THREADED versions of the large image benchmark on a four-processor, 250 MHz DEC Alpha running OSF.

The benchmark consists of the following steps:

Step	Task	Input [MB]	Output [MB]
1	Convert a 7761x7753 raw byte image to a 7761x7761 LLVS byte image.	60	60
2	Perform a 1D Gaussian on the byte image producing a float image.	60	240

3	Transpose the float image.	240	240
4	Perform a 1D Gaussian on the float image producing a float image.	240	240
5	Convert the float image to a byte image.	240	60
6	Transpose the byte image.	60	60

(LLVS is a low level image format developed at UMass)

It is evident from the results that the disk system on the DEC Alpha pre-fetches disk blocks (8k) and buffers at least 60 MB of data. This is evident from the IO counts from one step of the benchmark to the next.

The most significant factor effecting the elapsed time of the benchmark is the residency of the files. Placing the input file on a separate disk from the output file nearly halves the elapsed time. We believe that this is due to disk-head positioning as the data bandwidth to disk does not appear to be high enough to saturate the disk channel.

The elapsed time for the THREADED version is sometimes slightly lower than the elapsed time for the SYNCHRONOUS version regardless of file residency.

The CPU time for the THREADED version is always larger than the CPU time for the SYNCHRONOUS version. The difference is a function of the number of reads and writes and is inversely proportional to the block size read or written. The difference is due to the overhead of using pthreads, mutexes, and semaphores in order to implement asynchronous IO using pthreads.

In particular, the transpose operation is significantly more efficient using the SYNCHRONOUS version of the benchmark because the block size used for this operation is so small (597 bytes). (The transpose is accomplished by transposing sub-images. These sub-images are square and thus their dimension must be evenly divisible into both dimensions of the original image. Using a small sub-image reduces paging.) The block size for all other operations is a single row of the image.

Under these conditions, using the THREADED version of the benchmark is not productive. The disk system pre-fetch negates the benefit of the asynchronous IO and the overhead of using the POSIX pthreads significantly increases the CPU usage. Attached is the result from four sample runs on the Large Image benchmark. There are four sets of times:

1: THREADED - input & output to same disk
2: THREADED - input & output to different disks
3: SYNCHRONOUS - input & output to same disk
4: SYNCHRONOUS - input & output to different disks

Note that both the THREADED and the SYNCHRONOUS versions use the identical object modules for performing the various image processing operations. The only difference is in the ACCESS layer that is between these modules and the file system. This layer provides buffering. And, in the case of the THREADED version, allows overlapped IO with other processing (asynchronous IO).

The CPU times are in seconds. The elapsed times are in minutes. The Efficiency is the CPU time (user + system) divided by the elapsed time.

The Input & Output Ops. is the count of the number of 8k blocks transferred between the disk and RAM. (Note that 7350x8192 is 60,211,200 and 29460x8192 is 241,336,320.) The differences in counts and page faults is due to other activities being performed by the system for other users. Note that the input count for step two is either 0 or 1 in all four cases. We assume that this is due to the entire 60 MB image output from step one residing in either a operating system RAM buffer or a disk RAM buffer. The other counts show a similar pattern.

THREADED - - same disk

Step	Task	User Sec.	System Sec.	Elapsed Min.	Eff	Input Ops	Output Ops	Page Faults
1	Convert to square	12.90	10.40	0:15	146%	7351	7380	5
2	1D Gaussian	64.99	18.82	1:02	135%	1	29462	0
3	2D Transpose	60.47	49.05	5:00	36%	25260	29498	5
4	1D Gaussian	66.17	22.16	2:21	62%	13920	29463	18
5	Convert to byte	22.63	17.04	0:30	128%	15676	7371	0
6	2D Transpose	52.06	40.07	1:07	136%	2	7116	1
	Total	279.22	157.54	10:15		62210	110290	

THREADED - - different disk

Step	Task	User Sec.	System Sec.	Elapsed Min.	Eff	Input Ops	Output Ops	Page Faults
1	Convert to square	12.86	10.37	0:12	188%	7409	7371	8
2	1D Gaussian	64.73	19.16	1:02	135%	0	29468	0
3	2D Transpose	53.68	48.19	3:40	46%	23817	29493	1
4	1D Gaussian	66.38	24.00	1:24	107%	13825	29450	19
5	Convert to byte	22.58	17.36	0:22	175%	15882	7367	0
6	2D Transpose	41.95	38.18	1:00	131%	2	7120	1
	Total	262.18	157.26	7:40		60935	110269	

SYNCHRONOUS - - same disk

Step	Task	User Sec.	System Sec.	Elapsed Min.	Eff	Input Ops	Output Ops	Page Faults
1	Convert to square	3.78	3.46	0:15	47%	7346	7373	0
2	1D Gaussian	57.72	7.96	1:06	99%	0	29465	0
3	2D Transpose	18.06	32.50	4:42	17%	24063	29514	0
4	1D Gaussian	54.74	12.20	2:50	39%	12602	29468	6
5	Convert to byte	7.50	6.70	0:28	49%	15103	7367	0
6	2D Transpose	12.10	15.42	0:43	62%	0	7117	0
	Total	153.90	78.24	10:04		59114	110304	

Step	Task	User	System	Elapsed	Eff	Input	Output	Page

		Sec.	Sec.	Min.		Ops	Ops	Faults
1	Convert to square	3.27	3.21	0:12	52%	7346	7371	0
2	1D Gaussian	57.42	7.19	1:04	99%	1	29469	0
3	2D Transpose	18.13	31.77	3:22	24%	20962	29488	2
4	1D Gaussian	54.33	11.92	1:32	71%	14499	29468	7
5	Convert to byte	7.43	6.86	0:21	65%	15410	7365	0
6	2D Transpose	11.88	15.90	0:35	79%	0	7123	0
	Total	152.46	76.85	7:06		58218	110284	

Future Directions

1. Complete the ASYNCHRONOUS version of the ACCESS part and obtain timings for it.

2. Run the benchmark on other machines. It is doubtful if this will show anything interesting concerning the SYNCHRONOUS versus THREADED versions. Some systems may have lower mutex costs than the Alpha. Other systems may not do as much buffering. But, the first won't change the main result and the second just shows that the vendor did not do a good job.

3. Use multi-tasking for the image processing kernels. In this effort it would be better to change the interface to the buffering layer as the pre-reading does not seem to be beneficial and to set up for pre-reading has some cost. Even so, these kernels are obviously IO bound so it seems unlikely that significant improvement in elapsed time will occur from multi-tasking.

4. Package up the benchmark. We should discus if there is any valid reason to leave in the possibility of doing pre-reading. Some vendors may have some special capabilities that could use it. Certainly, the image processing kernels will be easier to understand if this capability is removed from the buffering interface.